

Formation R

Benoît Lepage

2025-07-31

Contents

1	Bienvenue sur cette formation au logiciel R	5
1.1	Pourquoi choisir R ?	5
1.2	Téléchargez le logiciel R	6
1.3	Téléchargez un IDE (RStudio recommandé)	7
1.4	Trouver de l'aide sur R	11
1.5	Conventions d'écriture	13
2	Les objets dans R	15
2.1	Manipuler les objets dans l'environnement	15
2.2	Principaux types de données	16
2.3	Principales structures de données	18
2.4	Objet à une seule valeur (scalaire ou texte)	18
2.5	Les vecteurs <code>vector()</code>	25
2.6	Les listes <code>list()</code>	32
2.7	Les matrices <code>matrix()</code>	33
2.8	Les bases de données <code>data.frames()</code>	33
2.9	Les objets R peuvent posséder des attributs <code>attributes()</code>	33

Chapter 1

Bienvenue sur cette formation au logiciel R

R est un logiciel accessible gratuitement permettant de réaliser des analyses statistiques dans un environnement windows, macOS ou Linux.

1.1 Pourquoi choisir R ?

Le logiciel est gratuit, très complet, avec une communauté d'utilisateurs très active dans le monde entier. Il est fréquent que les nouvelles méthodes d'analyses statistiques développées dans les équipes académiques soient d'abord mises à disposition sur R.

Le logiciel R repose sur l'utilisation de **scripts** dans lesquels nous allons **programmer** les analyses statistiques. Cette écriture sous forme de programmation peut paraître austère à première vue, mais est indispensable pour permettre la **reproductibilité** et la **transparence** des analyses. La même démarche de programmation est utilisée dans tous les logiciels statistiques professionnels (Stata, SAS, Python, Matlab, etc).

Pour utiliser R, les premières choses à faire sont de :

- télécharger le logiciel R
- et télécharger un Environnement de Développement Intégré (IDE) comme RStudio.

1.2 Téléchargez le logiciel R

Vous pouvez télécharger la dernière version stable du logiciel R sur le site du R project.

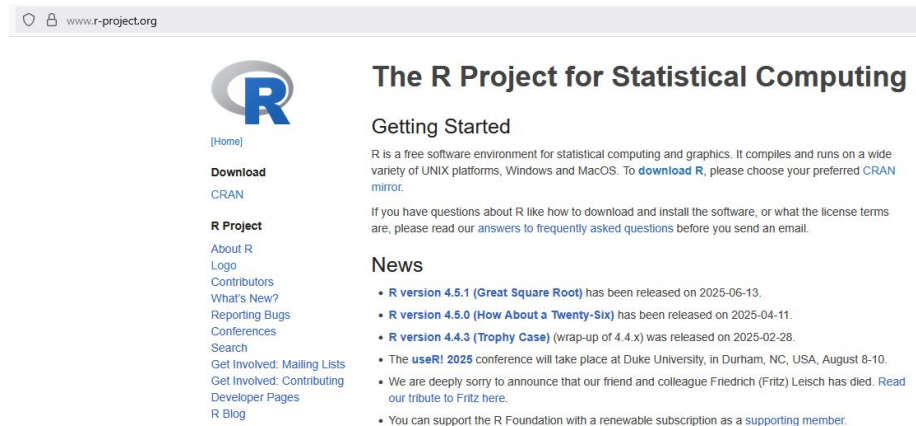


Figure 1.1: Site du R project, en juillet 2025

Cliquez sur “download R”, choisissez un site miroir (par exemple un des sites en France).

Puis téléchargez la version de R en fonction de votre système d’exploitation (Windows, macOS ou Linux).

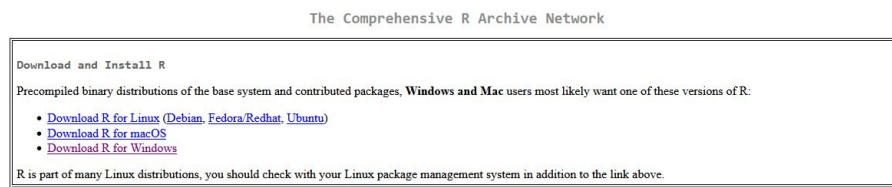


Figure 1.2: Choisissez la version adaptée à votre système d’exploitation

Enfin, installez R à partir du fichier d’installation que vous venez de télécharger.

1.2.1 Ouvrez le logiciel R

Si vous ouvrez le logiciel R, vous aller trouver l’interface graphique de R (*RGui* pour *R Graphical user interface*). Il est possible de faire vos analyses statistiques à partir de cette interface graphique, mais elle est très très austère.

Plutôt que d’utiliser cette interface RGui, nous vous recommandons fortement d’utiliser un Environnement de Développement Intégré (IDE), comme RStudio,

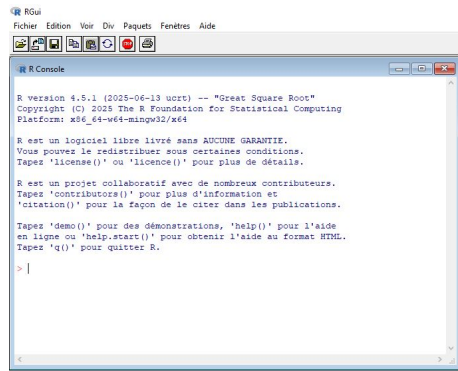


Figure 1.3: L'interface graphique de R (RGui)

qui vous facilitera grandement la vie pour utiliser un logiciel statistique qui repose sur de la programmation.

1.3 Téléchargez un IDE (RStudio recommandé)

RStudio est un environnement qui permet d'utiliser R, mais également d'autres logiciels de programmation comme Python, SQL, Stan, C++, etc. Cet environnement vous facilitera le travail pour :

- éditer vos scripts de programmation,
- accéder à la console,
- visualiser vos environnements de travail avec les fichiers et les objets qu'il contient,
- visualiser vos sorties graphiques et certaines tables d'analyses,
- visualiser vos données,
- visualiser les fichiers d'aide,
- gérer les *packages* permettant de faire des analyses spécifiques,
- et bien d'autres choses encore.

Par exemple, le tutoriel que vous êtes en train de lire a été créé à partir du package `bookdown` avec le logiciels R, au sein de l'IDE RStudio,

Vous pouvez télécharger la dernière version de RStudio sur le site de la compagnie Posit. Choisissez la version qui est adaptée à votre système d'exploitation (Windows, macOS ou Linux).

Puis, installez RStudio à partir du fichier d'installation que vous venez de télécharger.

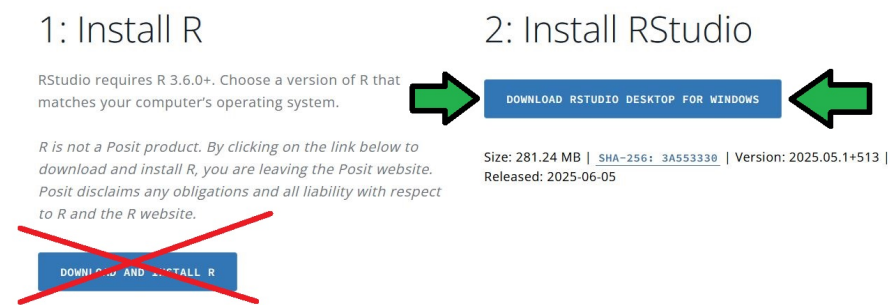


Figure 1.4: téléchargez RStudio

OS	Download	Size	SHA-256
Windows 10/11	RSTUDIO-2025.05.1-513.EXE	281.24 MB	3A553330
macOS 13+	RSTUDIO-2025.05.1-513.DMG	607.30 MB	76E15388
Ubuntu 22/Debian 12	RSTUDIO-2025.05.1-513-AMD64.DEB	209.78 MB	89A68B37
Ubuntu 24	RSTUDIO-2025.05.1-513-AMD64.DEB	209.78 MB	89A68B37
Fedora 41	RSTUDIO-2025.05.1-513-X86_64.RPM	224.98 MB	6A97DF24

Figure 1.5: téléchargez RStudio

1.3.1 Ouvrez l'IDE RStudio

Ouvrez RStudio, puis commencez par ouvrir un **script**

- à partir du menu File > New File > R script
- ou bien en utilisant le raccourci Ctrl+Maj+N sur windows
- ou bien en cliquant sur le petit fichier blanc avec un + vert en haut à gauche, puis choisir “R script”

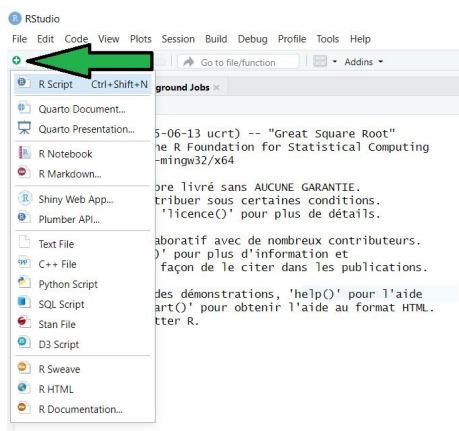


Figure 1.6: Ouvrir un nouveau script

L'interface de RStudio contient un menu, 4 quadrants et des sous-menus et boutons dans chaque quadrant.

Les menus qui vous seront le plus utiles sont :

- Dans le menu principal,
 - le menu *File* vous permettra de créer de nouveaux fichiers, d'ouvrir des fichiers déjà existants, de sauver vos fichiers, d'importer des bases de données, etc.
 - le menu *Tools* > *Install packages...* pour installer de nouveaux packages
 - le menu *Tools* > *Global Options...* vous permet de choisir la version du logiciel R à utiliser (onglet “R General”) ou bien de changer l'aspect graphique de l'environnement RStudio (onglet “Appearance”, puis choisissez un “Editor theme”, avec différentes interfaces claires ou sombres)
- Au sein du **script** (cadrant 1)
 - le bouton “disquette” permet de sauvegarder votre script

10CHAPTER 1. BIENVENUE SUR CETTE FORMATION AU LOGICIEL R

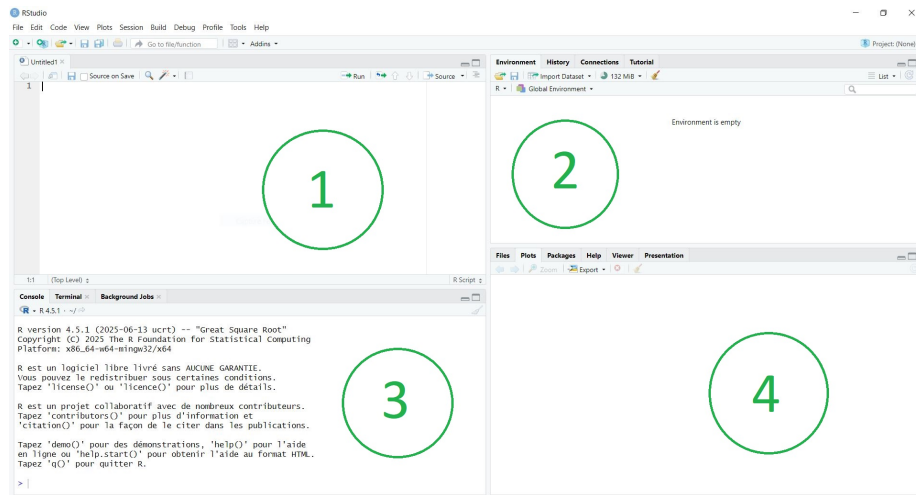


Figure 1.7: Les 4 cadrants de RStudio

- le bouton “run” permet de faire tourner votre programme d’analyse (les lignes que vous avez sélectionnées). Par exemple, tapez la commande suivante dans le script, sélectionnez la ligne et cliquez sur le bouton “run” (ou avec un raccourci clavier `ctrl+entrée` sur windows, ou encore `command+entrée` sur macOS).

```
print("Hello Toulouse")
```

et vous devriez voir la commande `> print("Hello Toulouse")` puis son résultat `"Hello Toulouse"` dans l’onglet **console** du cadrant 3.

- Au sein du cadrant 3, l’onglet le plus utile pour pour les débutants est l’onglet **console**
 - la console est la même que la console affichée dans l’interface RGui du logiciel R que l’on a vu au paragraphe 1.2.1.
 - la console commence par afficher la version de R en cours d’utilisation
 - vous pouvez y saisir des commandes et obtenir directement leurs résultats, par exemple si vous tapez dans la console `4+9`, vous obtiendrez directement le résultat 13. **Attention, les commandes que vous saisissez directement dans la console ne seront pas sauvegardées. Si vous voulez sauvegarder des commandes, il faut utiliser le *script* (cadrant 1)**

4+9

[1] 13

- Au sein du cadrant 2, l'onglet le plus utile pour les débutants est l'onglet **Environment**
 - cet onglet vous permettra de visualiser les “objets R” créés pendant vos analyses.
 - Par exemple si vous saisissez `v <- 1:10` dans la console, vous allez voir apparaître l'objet `v` dans l'environnement de travail (il s'agit d'un vecteur de 1 à 10, nommé “v”).
- Au sein du cadrant 4, les onglets les plus utiles pour les débutants sont :
 - l'onglet “File” qui contient les dossiers et fichiers au sein d'un dossier de travail (voir le chapitre 3 pour créer et organiser un dossier de travail associé à un “projet R”)
 - l'onglet “Plots” où vous retrouverez vos sorties graphiques. Au sein de cet onglet, vous trouverez un menu pour exporter vos graphiques selon différents formats. Des boutons permettent également de zoomer et d'effacer les graphiques. Par exemple, si vous saisissez `hist(rnorm(10000))` dans la console, un histogramme d'une distribution normale centrée réduite va apparaître. Vous pouvez effacer la figure en cliquant sur le bouton avec la croix rouge (efface la figure actuelle) ou le balet (efface l'ensemble des figures).
 - l'onglet “Packages” où vous pourrez activer, désactiver ou mettre à jour les packages qui ont été téléchargés.
 - l'onglet “Help” où vous trouverez de l'aide. Par exemple si vous saisissez `help(mean)` dans la console, l'aide de la commande `mean` va s'afficher. Vous pouvez également utiliser le champ de recherche de fonctions dans le menu “Help”.

1.4 Trouver de l'aide sur R

De nombreuses ressources sont disponibles pour vous aider à utiliser R :

- Les pages d'aide en ligne de R, qui apparaissent directement dans RStudio. Vous pouvez obtenir de l'aide sur des fonctions et des packages :
 - en appliquant une recherche par mot clé dans le champ de recherche de l'onglet “help”
 - en utilisant directement dans la console la fonction `help.search()` ou `??` associée à un mot clé (par exemple `help.search(student)` ou `??student`), ou la fonction `help()` associée à une fonction (par exemple `help(t.test)` ou `?t.test`)

- quand vous rencontrerez une nouvelle fonction (dans ce cours ou bien dans votre pratique), nous vous conseillons d’aller systématiquement regarder l’aide de cette fonction pour bien comprendre comment l’utiliser.
- Ces pages d’aide suivent la structure suivante :
 - * une partie “Description” qui décrit en quelques phrases ce que fait la fonction
 - * une partie “Usage” qui décrit la syntaxe de la fonction avec ses arguments
 - * une partie “Argument” qui précise comment renseigner les arguments de la fonction
 - * une partie “Detail” qui décrit en détail comment utiliser la fonction et ses arguments
 - * une partie “Value” qui décrit les sorties (les résultats) de la fonction, avec les éventuels sous-objets de la sortie
 - * une partie “Exemples” qui indique quelques exemple que vous pouvez directement lancer en cliquant sur “Run examples”
- Des fiches “mémoires” *cheat sheets* qui résument les principales fonctions :
 - pour les commandes R bases
 - des bonnes pratiques sur R
 - méthodes de visualisation avec le package ggplot2
 - l’utilisation du package data.table
 - l’utilisation du package dplyr et sa fiche dplyr
 - l’utilisation du package stringr pour manipuler les chaînes de caractères, avec la fiche stringr
 - la manipulation de dates Dates and times with lubridate et la fiche lubridate
- De nombreux livres et tutoriels disponibles gratuitement en ligne :
 - Le guide R de Joseph Larmarange, est un guide très complet et didactique, en français
 - L’Epidemiologist R Handbook est un tutoriel en anglais pour l’utilisation de R par des épidémiologistes
 - Software carpentry met à disposition des guides introductifs bien réalisés en anglais, par exemple R for Reproducible Scientific Analysis ou encore Programming with R
 - le livre R for Data Science propose une introduction très complète pour l’analyse de données descriptive principalement basée sur la suite de packages du Tidyverse
 - le package R swirl propose une formation interactive directement dans la console de RStudio, téléchargez le package (`install.packages("swirl")`), chargez le package (`library("swirl")`) et laissez vous guider après avoir saisi `swirl()` dans la console.

- RStudio Education liste plusieurs ressources intéressantes pour les débutants
- Le site CRAN a des manuels assez complets, par exemple la page R Language Definition est une introduction assez complète au langage de programmation R. La page CRAN Task Views liste les packages qui sont disponibles par thématique ou type d'analyse.
- Rechercher à l'aide d'un moteur de recherche (google, DuckDuckGo, Bing, etc). Ces recherches vous amèneront régulièrement vers des forums de discussion comme stackoverflow ou stackExchange. Si vous rencontrez une erreur ou une difficulté, il y a toutes les chances que d'autres personnes aient déjà rencontré ces erreurs et difficultés avant vous, et que des solutions détaillées soient proposées dans ces forums.
- Les chatbots de type GPT, Copilot ou Gemini : dans le cadre de votre formation au logiciel R, nous vous déconseillons l'utilisation de ces outils basés sur des LLM. Ces outils posent de nombreux problèmes en termes de transparence, de respect de droit d'auteur, d'impact environnemental, de déqualification (délégation de compétences pour rechercher de l'information, perte d'esprit critique), de dépendance aux Gafams, de dégradation des systèmes d'information, etc. Par ailleurs, bien qu'ils peuvent apporter des solutions fonctionnelles, il est bien plus utile d'avoir une bonne compréhension des bases de programmations sur R avant d'utiliser de tels outils : sans une bonne compréhension de la logique de programmation et des principales fonctions de R, vous aurez des difficultés à évaluer la fiabilité des solutions proposées, à vous débloquer en cas de problèmes, ou encore à adapter vos prompts pour obtenir de meilleures réponses. Les ressources décrites précédemment devraient vous permettre d'apporter efficacement des réponses à vos questions.

1.5 Conventions d'écriture

Certaines conventions ont été proposées pour faciliter l'écriture et la lecture du code de programmation dans R. Elles ne sont pas obligatoires, mais nous vous encourageons à les suivre.

Par exemple, le *Tidyverse style guide* :

- nommer les variables et les fonctions en lettres minuscules, à l'aide de mots et de chiffres séparés par `_` (*underscore*). Par exemple `csp_1`. Cette convention fait référence au style "snake case"
- ajouter un espace après une virgule, par exemple `x[2, 5]`
- ajouter des espaces avant et après les opérateurs arithmétiques, par exemple `x <- (1 + 2) / 5`, à quelques exceptions près (pas d'espace avant ou après le signe "puissance" `^`) `y <- x^2 + 3`

Chapter 2

Les objets dans R

2.1 Manipuler les objets dans l'environnement

La programmation R repose sur des *objets*, qui apparaîtront dans la fenêtre *Environment* de RStudio.

Voici quelques commandes de gestion des objets dans votre environnement :

- dans la console, commencez par créer les objets suivants. Pour **assigner une ou plusieurs valeurs à un objet**, on utilise une flèche dirigée vers la gauche <-. Vous verrez apparaître ces objets dans la fenêtre *Environment*.

```
# note : le signe dièse (#) permet d'ajouter des commentaires dans le code
# - le 1er objet est un vecteur de 10 entiers de 1 à 10
# - le 2ème objet est un vecteur de 3 lettres A, B et C
# - le 3ème objet est un vecteur de 2 réels, calculés par 2 opérations
# - le 4ème objet est une fonction qui ajoute 2 aux éléments du vecteur x
# - le 5ème objet est un scalaire égal à 42
objet_1 <- c(1:10)
objet_2 <- c("A","B","C")
objet_3 <- c(10 / 3, 4 * 5)
objet_4 <- function(x) {x + 2}
objet_5 <- 42
```

- la commande `ls()` permet de **lister** les objets dans l'environnement.
- la commande `rm()` permet de **supprimer** (*remove*) un ou plusieurs objets de l'environnement.

```
ls()
```

```
## [1] "objet_1" "objet_2" "objet_3" "objet_4" "objet_5"
```

```
rm(objet_2,objet_5)
```

```
rm(list = ls()) # pour supprimer tous les objets présents dans l'environnement
```

Conseils pour nommer un objet dans R :

- Utiliser des noms courts, mais lisibles, faciles à comprendre
- Les noms doivent commencer par une lettre (pas par un nombre) et ne peuvent pas contenir d'espace. A noter qu'ils peuvent contenir des points, comme par exemple `nom.variable`.
- Evitez d'utiliser des noms de variables et de fonctions déjà existants dans R (par exemple `mean` qui risque de porter à confusion avec la fonction `mean()` : il vaut mieux utiliser `mean_variable`)
- Ecrire en minuscule avec underscore pour séparer les mots (par exemple `date_naissance`)

2.2 Principaux types de données

Les données peuvent être de différents types :

- des **nombres** (`?numeric`). Ces nombres peuvent être des **nombres réels** (`?double`), par exemple 12.43. Il peut également s'agir de **nombres entiers** (`?integer`). Les nombres entiers sont saisis en ajoutant L à droite du nombre, par exemple 5L.
- des **chaînes de caractères textuels** (`?character`), définis avec des guillemets simples ou doubles, par exemple 'bonjour' ou "au revoir"
- des **valeurs logiques** (`?logical`), avec deux valeurs possibles :
 - valeur booléenne *vraie*, notée TRUE ou bien T
 - valeur booléenne *fausse*, notée FALSE ou bien F
- des **variables qualitatives nominales** (`?factor`) ou **ordinales** (`?ordered`)
- des **dates** (`?Date`)
- etc.
- une **valeur manquante** se note NA. L'**ensemble vide** se note NULL

2.2.1 Décrire le type de l'objet ♠

On peut décrire quel est le type de l'objet avec les fonctions `typeof` (le type le plus élémentaire), `mode` et `storage.mode` (type de l'objet et mode de stockage de l'objet selon un regroupement un peu plus large)

```
# les valeurs réelles ('double') et les entiers ('integer') sont de mode 'numeric'
typeof(2.53) # un réel
typeof(5L)  # et un entier
mode(2.53)  # sont de type 'numeric'
mode(5L)
storage.mode(2.53)
storage.mode(5L)

# les chaînes de caractères sont de type et de mode 'character'
typeof(c("hello", "Toulouse"))
mode(c("hello", "Toulouse"))

# les valeurs logiques sont de type 'logical'
typeof(c(TRUE, FALSE, FALSE))
mode(c(TRUE, FALSE, FALSE))
```

x	typeof(x)	mode(x)	storage.mode(x)
2.53	"double"	"numeric"	"double"
5L	"integer"	"numeric"	"integer"
"bonjour"	"character"	"character"	"character"
TRUE	"logical"	"logical"	"logical"
as.Date("2025-07-01")	"double"	"numeric"	"double"

Les fonctions `as.numeric`, `as.integer`, `as.character`, `as.logical` permettent de définir un objet *en tant que* numérique, entier, chaîne de caractères, logique.

```
as.numeric(5L) # définit un nombre entier en tant que nombre réel
as.integer(4.95) # définit un réel en tant qu'entier, seul l'entier est conservé
as.character(4.95) # définit un nombre en tant que chaîne de caractères

# définir une valeur logique TRUE et FALSE en tant que valeur numérique
# ou en tant qu'entier donne les valeurs 1 et 0, respectivement
as.numeric(TRUE)
as.numeric(FALSE)

# définir le nombre 0 en tant que valeur logique donne la valeur FALSE
```

```
as.logical(0)

# définir tout nombre différent de 0 en tant que valeur logique
# donne la valeur TRUE
as.logical(-14)
as.logical(1)
as.logical(4.95)
```

Les fonctions `is.numeric`, `is.integer`, `is.character`, `is.logical` permettent d'évaluer si un objet est de type numérique, entier, textuel, logique.

```
is.numeric(5L) # TRUE, un entier est bien un objet numérique
is.integer(4.95) # FALSE, 4.95 n'est pas un entier
is.numeric("bonjour") # FALSE "bonjour" est une chaîne de caractères
is.character("bonjour") # TRUE, "bonjour" est bien une chaîne de caractères
is.character(4.95) # FALSE, 4.95 est un objet numérique
is.logical(1) # FALSE, 1 est un objet numérique
is.logical(as.logical(1)) # TRUE, as.logical(1) = TRUE, qui est un objet logique
is.logical(TRUE) # TRUE est bien un objet logique
```

2.3 Principales structures de données

Les principales structures de données que nous allons détailler dans la suite de ce chapitre sont :

- les vecteurs (`?vector`), dont font partie les scalaires (vecteurs à une seule valeur).
- les matrices (`?matrix`, `?array`)
- les listes (`?list`)
- les bases de données (`?data.frames`). Il existe d'autres formats de base de données qui seront présentés plus tard (avec les packages `tidyverse` et `data.table` par exemple)

2.4 Objet à une seule valeur (scalaire ou texte)

2.4.1 Scalaires

Assignez les valeurs 4 et 5 à deux objets

```
x_1 <- 4
x_2 <- 5
```

2.4.2 Opérations mathématiques sur les scalaires

2.4.2.1 Calculatrice

On peut utiliser les opérations classiques, comme sur une calculatrice :

- + pour **additionner**
- - pour **soustraire**
- * pour **multiplier**
- / pour **diviser**
- ^ pour mettre à la **puissance**
- e pour la **notation scientifique**

```
x_1 + x_2 # 4 + 5 = 9
10 - x_1 # 10 - 4 = 6
x_1 * x_2 # 4 * 5 = 20
20 / x_2 # 20 / 5 = 4
x_1^2 # 4^2 = 16
10^-1 # 1/10 = 0.1
25^(0.5) # racine carrée de 25 (puissance 1/2)

# notation scientifique pour les grands et petits nombres
1/1000000 # 1 pour 1 million = 1e-6
1/1e6
1e6 * 1000 # 1 million * 1000 = 1 milliard
```

2.4.2.2 Fonctions mathématiques

Plusieurs fonctions mathématiques de bases sont implémentées nativement dans R :

- `log(x)` ou `log(x, base = exp(1))` pour le **logarithme** népérien,
- `log10(x)` pour le logarithme base 10, `log2(x)` pour le logarithme base 2,
- `log(x, base = b)` pour le logarithme base `b`,
- `exp(x)` pour l'**exponentielle** de `x`
- `sqrt(x)` pour la **racine carrée** de `x`
- `abs(x)` pour la **valeur absolue** de `x`
- les **fonctions trigonométriques** sont implémentées, avec `cos(x)`, `sin(x)`, `tan(x)` (cf. `?Trig`)
- la **constante** π est implémentée avec `pi` (cf. `?Constants`)

Si vous appliquez une fonction à une valeur qui ne fait pas partie du domaine de définition de la fonction, le résultat sera une valeur manquante notée **NaN** (*not a number*). Un message d'avertissement va apparaître si vous appliquez une fonction en dehors de son domaine de définition.

```
# logarithmes et exponentielles
log(1)
log10(100)
log(100, base = 10)
exp(1)

# racine carrée
sqrt(x_2^2)

# valeur absolue
abs(10)
abs(-10)

# fonctions trigonométriques
cos(1)
sin(1)
tan(1)
pi
2 * pi * 10 # circonférence d'un cercle de rayon 10

# si on utilise une valeur en dehors du domaine d'application de la fonction
log(-1) # -1 est en dehors du domaine de définition de la fonction log
sqrt(-2) # -2 est en dehors du domaine de définition de la fonction racine carrée
```

2.4.2.3 Fonctions d'arrondi ♠

Plusieurs fonctions sont disponibles dans R pour arrondir une valeur (cf. ?Round) :

- la fonction `round()` est utile pour arrondir les décimales. Il faut préciser en argument, le nombre de chiffres après la virgule. **Attention** : si le nombre se termine par un 5, l'arrondi se fait vers le chiffre pair le plus proche : 4.45 s'arrondit à 4.4 et 4.75 s'arrondit à 4.8
- la fonction `signif()` arrondit aux chiffres les plus significatifs (les plus grands)
- la fonction `floor()` arrondit la valeur à l'entier inférieur
- la fonction `ceiling()` arrondit la valeur à l'entier supérieur
- la fonction `trunc()`

```
## fonction round()
# l'argument digits permet de définir le nombre de chiffres après la virgule
# exemple si vous voulez arrondir à 2 chiffres après la virgule
round(0.09400, digits = 2)
round(0.08600, digits = 2)
# arrondir à 1 chiffre après la virgule
round(4.450, digits = 1)
round(4.750, digits = 1)
# pour arrondir une valeur 5, le résultat va vers le chiffre pair le plus proche
round(4.5, digits = 0) # arrondi à l'entier pair le plus proche
round(1.5, digits = 0) # arrondi à l'entier pair le plus proche

## fonction signif()
# on garde les valeurs les plus significative, définie par l'argument digits
signif(123.456789, digits = 1)
signif(123.456789, digits = 2)
signif(123.456789, digits = 3)
signif(123.456789, digits = 4)
signif(123.456789, digits = 5)
# la règle d'arrondi vers le chiffre pair le plus proche est également appliquée
signif(4.45, digits = 3)
signif(4.45, digits = 2)
signif(4.75, digits = 3)
signif(4.75, digits = 2)

## fonction trunc() supprime simplement les décimales
# note : ici, il n'y a pas d'arrondi vers la chiffre pair la plus proche)
trunc(123.456)
trunc(4.5)
trunc(1.5)

## la fonction floor() arrondit à l'entier inférieur
floor(4.1)
floor(4.9)

## la fonction ceiling() arrondit à l'entier supérieur
ceiling(4.1)
ceiling(4.9)
```

2.4.3 Concaténation de chaînes de caractères

On peut concatener deux objets en chaînes de caractères :

- la fonction `paste()` concatène les chaînes de caractères en séparant les

vecteurs par un espace (argument par défaut, cf `?paste`). Cet argument peut être modifié.

- la fonction `paste0()` concatène les chaînes de caractères sans espace.

```
x1 <- "Bonjour"
x2 <- "Toulouse"
paste(x1, x2)
```

```
## [1] "Bonjour Toulouse"
```

```
paste0(x1, x2)
```

```
## [1] "BonjourToulouse"
```

```
paste(x1, x2, sep = ", ") # ici on sépare x1 et x2 par une virgule et un espace
```

```
## [1] "Bonjour, Toulouse"
```

```
# vous pouvez inclure des nombres qui seront transformés en caractères
paste0(x1, 123, x2)
```

```
## [1] "Bonjour123Toulouse"
```

2.4.4 Valeurs logiques TRUE et FALSE

2.4.4.1 Evaluer des conditions

Nous pouvons utiliser les opérateurs de comparaison ci-dessous pour évaluer des conditions :

- `==` ... est égal à ...
- `!=` ... est différent de ...
- `<` ... est inférieur à ...
- `>` ... est supérieur à ...
- `<=` ... est inférieur ou égal à ...
- `>=` ... est supérieur ou égal à ...
- `%in%` ... est inclus dans ...

Par exemple, nous pouvons évaluer les comparaisons suivantes, la réponse attendue est vraie (TRUE) ou fausse (FALSE).

```

5 == 10 # est-ce que 5 est égal à 10 ?
5 != 10 # est-ce que 5 est différent de 10 ?
5 < 10 # est-ce que 5 est inférieur à 10 ?
5 > 10 # est-ce que 5 est supérieur à 10 ?
5 <= 5 # est-ce que 5 est inférieur ou égal à 5 ?
5 >= 5 # est-ce que 5 est supérieur ou égal à 5 ?
5 %in% c(4,5,6) # est-ce que 5 est inclus dans le vecteur (4,5,6) ?
5 %in% c(7,8,9) # est-ce que 5 est inclus dans le vecteur (7,8,9) ?

```

La fonction `identical` permet d'évaluer si deux objets sont exactement égaux. Elle peut s'appliquer à des valeurs simples mais aussi à des objets de plus grandes dimensions (vecteurs, matrices, bases de données, ...)

```

identical(5, 10) # équivalent à la commande 5 == 10
identical(c(1,2,3), c(1,2,3)) # les deux vecteurs (1,2,3) sont bien les mêmes

```

Une comparaison à une valeur manquante (NA) retournera une valeur manquante.

Attention, si vous souhaitez évaluer si une valeur est manquante, il faut utiliser la fonction `is.na(x)` (plutôt que `x == NA` qui est déconseillé).

```

NA < 10
is.na(10) # éviter d'utiliser 10 == NA pour tester si une valeur est manquante
is.na(NA)
is.na(c(1,2,3,NA,5,6,NA,8,9,10))

```

2.4.5 Opérations sur des valeurs logiques

On peut combiner des valeurs logiques avec les opérateurs logiques ET, OU, et NON (négation logique)

- `&` opérateur ET
- `|` opérateur OU (sur windows, combinaison de touches altgr + 6 ; sur macOS, combinaison de touche alt + maj + L)
- `!` opérateur NON (négation logique : “n’est pas”)

Les résultats attendus d'une combinaison d'opérateurs logiques sont résumés dans les table de vérité ci-dessous.

- Opérateur ET

a	b	a ET b
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

- Opérateur OU

a	b	a OU b
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

- Opérateur NON

a	NON a
TRUE	FALSE
FALSE	TRUE

```
# opérateur ET
TRUE & TRUE
TRUE & FALSE
FALSE & TRUE
FALSE & FALSE
(5 > 10) & (2 != 5) # TRUE ET TRUE donne TRUE
(5 > 10) & (2 == 5) # TRUE ET FALSE donne FALSE
(5 < 10) & (2 == 5) # FALSE ET FALSE donne FALSE

# opérateur OU
TRUE | TRUE
TRUE | FALSE
FALSE | TRUE
FALSE | FALSE
(5 < 10) | (2 != 5) # TRUE OU TRUE donne TRUE
(5 > 10) | (2 != 5) # FALSE OU TRUE donne TRUE
(5 > 10) | (2 == 5) # FALSE OU FALSE donne FALSE

# opérateur NON
!TRUE
```



```
!FALSE
!(5 < 10) # non-TRUE donne FALSE
!(5 > 10) # non-FALSE donne TRUE
```

Il existe également un opérateur `xor()` correspondant au OU EXCLUSIF :

- Opérateur OU EXCLUSIF

a	b	a OU EXCLUSIF b
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

```
# opérateur OU EXCLUSIF
xor(TRUE, TRUE)
xor(TRUE, FALSE)
xor(FALSE, TRUE)
xor(FALSE, FALSE)
xor((5 < 10), (2 != 5)) # (TRUE) OU exclusif (TRUE) donne FALSE
xor((5 > 10), (2 != 5)) # (FALSE) OU exclusif (TRUE) donne TRUE
xor((2 != 5), (5 > 10)) # (TRUE) OU exclusif (FALSE) donne TRUE
xor((5 > 10), (2 == 5)) # (FALSE) OU exclusif (FALSE) donne FALSE
```

2.5 Les vecteurs `vector()`

2.5.1 Création d'un vecteur de valeurs

La fonction `c()` (pour concaténer) permet de créer des vecteurs de valeurs. Ces valeurs doivent toutes être du même type et séparées par une virgule :

```
# Un vecteur de 3 nombres réels
vect_num <- c(1, pi, 3.54)
print(vect_num) # [1] 1.000000 3.141593 3.540000

# Un vecteur de 5 entiers
vect_int <- c(5L, 4L, -3L, 15L, -8L)
print(vect_int) # [1] 5 4 -3 15 -8

# Si on combine ces 2 vecteurs, on obtient un vecteur de 8 nombres réels
```

```

# (les nombres entiers sont automatiquement transformés en nombres réels pour
# qu'ils soient tous du même type)
vect_new <- c(vect_num, vect_int)
print(vect_new)
# 1.000000 3.141593 3.540000 5.000000 4.000000 -3.000000 15.000000 -8.000000

# Un vecteur de 6 chaînes de caractères
vect_char <- c("a", "B", "XYZ", "HELLO", "HOMME", "femme")
vect_char # [1] "a"      "B"      "XYZ"    "HELLO"  "HOMME"  "femme"

# Un vecteur logique
# pour les valeurs logiques, on peut utiliser les abréviations T pour TRUE,
# et F pour FALSE
vect_logic <- c(TRUE, FALSE, FALSE, F, T, T, FALSE, TRUE)
print(vect_logic) # [1] TRUE FALSE FALSE FALSE TRUE TRUE FALSE TRUE

# la fonction class() permet de décrire le type du vecteur
class(vect_num) # numeric
class(vect_int) # vect_int
class(vect_new) # numeric
class(vect_char) # character
class(vect_logic) # logical

# Si vous créez le vecteur suivant en mélangeant valeurs numériques et
# chaînes de caractères
vect_melange <- c(1,2,"trois",4)
# les valeurs numériques sont transformées de force en chaînes de caractères
# c("X","X",...,"X","X") pour que toutes les valeurs soient du même type.
print(vect_melange)
# [1] "1"      "2"      "trois"  "4"

```

2.5.1.1 Vecteur d’une variable qualitative

Les variables qualitatives sont de type **factor** (pour des variables qualitatives nominales) ou de type **ordered** (pour les variables qualitatives ordinales).

Par exemple, on va partir d’une variable codée avec des nombres que l’on souhaite interpréter comme les catégories d’une variable qualitative : “0 = non”, “1 = oui”, “9 = ne sais pas”.

Les fonctions **factor()** et **as.factor()** permettent de définir une variable qualitative nominale. De même, les fonctions **ordered()** et **as.ordered()** permettent de définir une variable qualitative ordinale.

Les objets de type **factor** (ou **ordered**) ont une liste d’**attributs** que l’on peut obtenir par la fonction **attributes()**. Par défaut, les fonctions **factor()** et

`ordered()` vont créer des `levels`, correspondant aux catégories de la variable qualitative. Pour cela, il va utiliser chaque valeur unique, les ranger de la plus petite à la plus grande, et transformer ces valeurs en chaînes de caractères.

```
# On commence avec la variable saisie sous forme de vecteur d'entiers
var_1 <- c(0, 9, 0, 0, 1, 9, 1, 0, 0, 9, 1, 9, 0, 1, 1)

# vous pouvez définir ce vecteur numérique en "factor"
var_quali <- as.factor(var_1) # autre possibilité : var_quali <- factor(var_1)
print(var_quali)
class(var_quali)

# les fonction "is.factor" ou "is.ordered" permettent de vérifier si le type de
# l'objet est bien une variable qualitative nominale ("factor") ou ordinale
# ("ordered")
is.factor(var_1) # FALSE, l'objet "var_1" est un vecteur numérique
is.factor(var_quali) # TRUE, l'objet "var_quali" est bien de type "factor"

# La variable var_quali a des attributs, que l'on va enregistrer dans un nouvel
# objet "attr_var_quali"
attr_var_quali <- attributes(var_quali)
class(attr_var_quali) # les attributs sont un objet de type "list"
attr_var_quali
# la liste contient deux sous-objets "levels" et "class".
# On peut obtenir les sous-objets de la liste avec l'opérateur dollar "$"
attr_var_quali$levels # "0" "1" "9"
attr_var_quali$class # factor

class(attr_var_quali$levels)
# les levels sont définis comme une chaîne de caractères, rangés du plus petit
# au plus grand
```

Alternativement, on peut attribuer des étiquettes `labels = c("non", "oui", "ne sait pas")` à la place des `levels = c("0", "1", "9")`. Attention, les étiquettes doivent être indiquées dans le même ordre que les `levels` (du code le plus petit au plus grand) (cf. `?factor`).

Si vous souhaitez retransformer la variable en format numérique, vous pouvez appliquer la fonction `as.numeric` au facteur. Toutefois, une nouvelle numérotation par défaut sera appliquée en commençant par la valeur 1.

```
# On reprend la variable précédente (avec de possibles valeurs manquantes NA)
var_1 <- c(0, 2, 0, 0, 1, 2, NA, 0, NA, 2, 1, 2, 0, 1, 1)

# on renseigne les arguments
var_quali <- factor(var_1,
```

```

                                labels = c("non", "oui", "ne sait pas"))
var_quali
# si l'on demande d'imprimer la variable, les valeurs apparaissent à présent
# sous leurs étiquettes

attr_var_quali <- attributes(var_quali)
attr_var_quali$levels
# les levels sont à présent le vecteur de caractère c("non","oui","ne sait pas")

new_var_quali <- as.numeric(var_quali)
new_var_quali
# [1]  1  3  1  1  2  3 NA  1 NA  3  2  3  1  2  2

# si vous voulez que les manquants soient considérés comme une catégorie
# en tant que telle, on peut utiliser la fonction "addNA"
addNA(var_quali, ifany = TRUE)

```

2.5.1.2 Fonction length()

Les vecteurs sont des objets à 1 seule dimension. On peut décrire le nombre de valeurs qu'il contient avec la commande `length()` (longueur du vecteur).

```

length(vect_num) # vecteur de longueur 3
length(vect_int) # vecteur de longueur 5
length(vect_new) # vecteur de longueur 8
length(vect_char) # vecteur de longueur 6
length(vect_logic) # vecteur de longueur 8
length(var_quali) # vecteur de longueur 15

```

2.5.1.3 Nommer (names()) les éléments d'un vecteur

Il est possible d'associer des noms à chaque éléments d'un vecteur, à l'aide d'un vecteur de chaînes de caractères. Ces noms seront attachés aux attributs du vecteur.

```

# on créer un vecteur de scores de satisfaction entre 1 et 5
# 1 = non satisfait
# 2 = un peu satisfait
# 3 = moyennement satisfait
# 4 = plutôt satisfait
# 5 = très satisfait
satisf <- c(1,4,2,3,1,3,2,5,4,5,3,2,3)

```

```
# on associe des noms aux valeurs 1,2,3,4,5
names(satisf) <- c("non","plutôt","un peu","moyen","non","moyen","un peu",
                  "très","plutôt","très,moyen","un peu","moyen")
```

2.5.1.4 Créer une séquence de valeurs

Pour créer une séquence de nombre entiers, ascendante ou descendante de 1 en 1, vous pouvez utiliser la commande :

```
vect_1a10 <- 1:10 # vecteur ascendant
vect_1a10 # [1] 1 2 3 4 5 6 7 8 9 10

vect_3_a_moins5 <- 3:-5 # vecteur descendant
vect_3_a_moins5 # [1] 3 2 1 0 -1 -2 -3 -4 -5
```

La commande `seq()` permet de définir des séquences de manière plus flexible, grâce aux arguments `from`, `to`, `by`, `length.out`, `along.with` (regardez l'aide `?seq()`).

```
seq(from = 1, to = 10) # vecteur de 1 à 10, équivalent à la commande 1:10
seq(from = 0, to = 10, by = 2) # vecteur de 0 à 10, de 2 en 2

seq(from = 1, to = 50, length.out = 5) # vecteur de 1 à 50, de longueur 5
# les intervalles entre les valeurs sont calculés automatiquement par la formule
# (valeur de départ - valeur d'arrivée) / (longueur totale - 1)

seq(from = 0, to = 60, along.with = c("dix","vingt","trente"))
# vecteur de 0 à 60, dont la longueur est égale à la longueur du vecteur
# c("dix","vingt","trente") (sa longueur est de 3)

# avec l'argument `by`, si le cycle ne tombe pas juste, la séquence s'arrête
# avant la dernière valeur indiquée par `to = `
seq(from = 1, to = 10, by = 2)
# [1] 1 3 5 7 9
# la séquence ne va pas jusqu'à 10, elle s'arrête à 9, car 9 + 2 = 11
# ce qui dépasserait la valeur maximale demandée par `to = 10`
```

2.5.1.5 Créer un vecteur de valeurs répétées

La fonction `rep` permet de créer des vecteurs de valeurs répétées. Les arguments `times`, `each`, `length.out` permettent de préciser comment les valeurs doivent être répétées (voir dans l'aide `?rep`)

```
rep(5, times = 3) # répète la valeur 5, 3 fois.
# [1] 5 5 5

rep(1:4, times = 3) # répète 3 fois le vecteur c(1, 2, 3, 4)
# [1] 1 2 3 4 1 2 3 4 1 2 3 4

rep(1:4, each = 3) # chaque élément du vecteur c(1, 2, 3, 4) est répété 3 fois
# [1] 1 1 1 2 2 2 3 3 3 4 4 4

rep(1:4, length.out = 10) # le vecteur c(1,2,3,4) est répété dans un vecteur
# dont la longueur totale est de 10
# [1] 1 2 3 4 1 2 3 4 1 2
```

2.5.2 Opérations arithmétiques sur un vecteur

On applique une opération arithmétique avec un scalaire (une valeur simple) à chacun des éléments d'un vecteur :

```
# additionne +2 à chaque élément de la séquence c(1,2,3,4,5)
seq(from = 1, to = 5) + 2 # cela donne 3 4 5 6 7

# soustrait -2 à chaque élément de la séquence c(1,2,3,4,5)
seq(from = 1, to = 5) - 2 # cela donne -1 0 1 2 3

# multiplie par 5 à chaque élément de la séquence c(1,2,3,4,5)
seq(from = 1, to = 5) * 5 # cela donne 5 10 15 20 25

# divise par 5 chaque élément de la séquence c(1,2,3,4,5)
seq(from = 1, to = 5) / 5 # cela donne 0.2 0.4 0.6 0.8 1.0
```

Si vous appliquez une opération arithmétique entre deux vecteurs de même longueur, l'opération se fera entre les 1ers éléments de chaque vecteur, puis entre les 2èmes éléments de chaque vecteur, puis etc

```
vec_A <- c(1, 2, 3, 4, 5)
vec_B <- c(-1, +1, -2, +2, -5)

# addition : c((1 + (-1)), (2 + 1), (3 + (-2)), (4 + 2), (5 + (-5)))
vec_A + vec_B # 0 3 1 6 0

# soustraction : c((1 - (-1)), (2 - 1), (3 - (-2)), (4 - 2), (5 - (-5)))
vec_A - vec_B # 2 1 5 2 10
```

```
# multiplication : c((1 * (-1)), (2 * 1), (3 * (-2)), (4 * 2), (5 * (-5)))
vec_A * vec_B # -1  2  -6  8 -25

# division : c((1 / (-1)), (2 / 1), (3 / (-2)), (4 / 2), (5 / (-5)))
vec_A / vec_B # -1.0  2.0 -1.5  2.0 -1.0
```

Si vous appliquez une opération arithmétique entre deux vecteurs de longueur différente, l'opération se fera entre les 1ers éléments de chaque vecteur, puis entre les 2èmes éléments de chaque vecteur, etc. Lorsqu'on arrive au bout du vecteur le plus court, les opérations continuent en reprenant à partir de la première valeur du vecteur le plus court. Un message d'avertissement vous prévient également lorsque la longueur du vecteur le plus long n'est pas un multiple de la longueur du vecteur le plus court (mais cela n'empêche pas l'opération de se faire).

```
vec_A <- c(1, 2, 3, 4, 5)
vec_C <- c(1, 2, 3)

# addition : c((1 + 1), (2 + 2), (3 + 3), (4 + 1), (5 + 2))
vec_A + vec_C # 2 4 6 5 7

# soustraction : c((1 - 1), (2 - 2), (3 - 3), (4 - 1), (5 - 2))
vec_A - vec_C # 0 0 0 3 3

# multiplication : c((1 * 1), (2 * 2), (3 * 3), (4 * 1), (5 * 2))
vec_A * vec_C # 1 4 9 4 10

# division : c((1 / 1), (2 / 2), (3 / 3), (4 / 1), (5 / 2))
vec_A / vec_C # 1.0 1.0 1.0 4.0 2.5
```

On peut combiner des vecteurs de chaînes de caractères à des vecteurs numériques :

```
paste0("L", seq(from = 1, to = 10))
# on obtient le vecteur :
# "L1" "L2" "L3" "L4" "L5" "L6" "L7" "L8" "L9" "L10"

paste0(c("A","B","C"), seq(from = 1, to = 10))
# on obtient le vecteur :
# "A1" "B2" "C3" "A4" "B5" "C6" "A7" "B8" "C9" "A10"
```

On peut faire des opérations sur des vecteurs logiques (voir les tables de vérité plus haut) :

```

c(1, 2, 3) %in% seq(from = 1, to = 9, by = 2)
# TRUE FALSE TRUE
# les valeurs 1 et 3 sont bien comprise dans le vecteur c(1, 3, 5, 7, 9),
# mais pas la valeur 2

seq(from = 1, to = 9, by = 2) %in% c(1, 2, 3)
# TRUE TRUE FALSE FALSE FALSE
# les valeurs 1 et 3 sont bien comprise dans le vecteur c(1, 2, 3),
# mais pas les valeurs 5, 7 et 9

c(TRUE, TRUE, FALSE, FALSE) & c(TRUE, FALSE, TRUE, FALSE)
# TRUE FALSE FALSE FALSE
# 1er élément avec le 1er élément TRUE & TRUE = TRUE
# 2ème élément avec le 2ème élément TRUE & FALSE = FALSE
# 3ème élément avec le 3ème élément FALSE & TRUE = FALSE
# 4ème élément avec le 4ème élément FALSE & FALSE = FALSE

c(TRUE, TRUE, FALSE, FALSE) | c(TRUE, FALSE, TRUE, FALSE)
# TRUE TRUE TRUE FALSE
# 1er élément avec le 1er élément TRUE | TRUE = TRUE
# 2ème élément avec le 2ème élément TRUE | FALSE = TRUE
# 3ème élément avec le 3ème élément FALSE | TRUE = TRUE
# 4ème élément avec le 4ème élément FALSE | FALSE = FALSE

```

2.5.3 Indicage d'un vecteur

Pour sélection des sous-ensembles de valeurs au sein d'un vecteur, R utilise un indicage pour sélectionner les éléments d'un vecteur

fonctions `which()`, `match()`, `any()`, `all()`

2.5.4 fonctions statistiques pour résumer une série de valeurs

`min()` `max()` `median()` `quantile()` `mean()` `sd()` `var()` `sum()` `prod()`
`table()` `prop.table()` `unique()` `sort()`

2.6 Les listes `list()`

2.6.1 Création d'une liste

Les vecteurs peuvent être reformatés sous forme de liste

2.6.2 Indigage d'une liste

2.7 Les matrices `matrix()`

2.7.1 Création d'une matrice

Les vecteurs peuvent être reformatés sous forme de matrice

2.7.2 Indigage d'une matrice

2.7.3 matrices à plus de 2 dimensions `array()`

2.8 Les bases de données `data.frames()`

2.9 Les objets R peuvent posséder des attributs `attributes()`